华东师范大学计算机科学与技术学院上机实践报告

课程名称：数据结构	**年级**：2022 级	**上机实践成绩**：
指导教师：金健	**姓名**：岳锦鹏	**上机实践时间**：2 学时
上机实践名称：第七章作业	**学号**：10213903403	**上机实践日期**：2023/12/15

一、 实验目的

1. 使用第七章的知识解决查找树的综合问题。

二、 实验内容

1. 给出 N 个正整数作为二叉排序树的节点插入顺序. 判断这串序列是否是该二叉树的先序序列或者是该二叉排序树的镜像树的先序序列. 镜像树指的是交换根结点的左右子树形成的二叉树，如果对于输入的序列与二叉排序树的先序序列一致，则输出 Yes，并输出该二叉排序树的后序序列. 若对于输入的序列与二叉排序树的镜像树的先序序列一致，则输出 Yes，并输出该二叉排序树的后序序列. 其余情况输出 No.
2. 根据二叉排序树的镜像树的后序遍历序列作为顺序，构造平衡二叉树。

三、 实验原理

1. 程序设计原理。

四、 实验步骤

1. 问题抽象
2. 编写程序
3. 调试程序
4. 完善总结

五、 调试过程、结果和分析

1. 树的结构确实不好调试，尤其是平衡二叉树，但好在可以进行中序遍历，正确的情况下中序遍历的结果应该是排好序的；
2. 对于镜像树，可以修改比较操作的函数定义，就像 C++ 内置的 make_heap 一样，当然由于镜像树是对称的，可以直接在遍历时先遍历右子树，再遍历左子树，也能实现；
3. 两道题的输入输出有关联性，第二题的输入是第一题的输出但又不完全是，第一题输出的是原二叉树的后序遍历结果，但第二题要使用第一题的镜像数的后序遍历结果，因此可以使用文件记录输出结果，第二题就可以进行输入重定向。

六、 总结

1. 遵循指针应在初始化时分配资源，以及将资源交给变量作用域和变量生命周期自动管理，此次代码中将每个节点的左右指针都使用 unique_ptr 定义，这样就不需要自己进行 delete 删除分配的资源了；
2. 但是使用 unique_ptr 要注意，此指针指向的对象不能再被其他对象指向，因此需要使用 move 进行转移资源的所有权，这在旋转子树时尤其明显；
3. 借助 std::function 和匿名函数，我们可以很方便地实现简单的函数回调，尤其是涉及到类和对象的成员函数的时候；
4. 类中的 this 是常量指针，是不能改变它所指向的地址的，即不能对 *this 赋值；
5. C++ 中有类似 Python 的迭代器，也能用匿名函数实现类似 JavaScript 的回调；
6. 但是 C++ 中没法直接使用 this->left_child && this->left_child->postorder_traversal(output)，因为逻辑与操作中的函数调用会返回 void，而 void 无法转换为 bool 类型；
7. C++ 中的模板自动类型推断，还有解包参数，和 Python、JavaScript 中有相似之处，但又不完全一样，仍需深入了解。

七、 附件

1. 第七章作业 1.cpp

```cpp
//
// Created by 423A35C7 on 2023-12-14.
//
// 20:30
// 22:20

#include <algorithm>
#include <fstream>
#include <functional>
#include <iostream>
#include <memory>
#include <vector>

template<typename T>
class BiSortNode {
    std::unique_ptr<BiSortNode> left_child;
    std::unique_ptr<BiSortNode> right_child;
    T data;
    std::function<bool(T&, T&)> compare_function_;

public:
    // 好像这里没法使用 initializer_list，initializer_list 是对于多个相同值的初始化，而不是一个东西的初始化参数列表
    template<typename... Args>
    explicit BiSortNode(Args... args, std::function<bool(T&, T&)> _compare) : data(args...) {
```

```
25          this->compare_function_(_compare);
26      }
27
28      template<typename... Args>
29      explicit BiSortNode(Args... args) : data(args...) {
30          this->compare_function_ = [](T&a, T&b) { return a < b; };
31      }
32
33      void insert(T new_data) {
34          if (new_data < this->data) {
35              if (this->left_child == nullptr) {
36                  this->left_child = std::make_unique<BiSortNode>(new_data);
37              }
38              else {
39                  this->left_child->insert(new_data);
40              }
41          }
42          else {
43              if (this->right_child == nullptr) {
44                  this->right_child =
                    ↪  std::make_unique<BiSortNode>(new_data);
45              }
46              else {
47                  this->right_child->insert(new_data);
48              }
49          }
50      }
51
52      void preorder_traversal(std::vector<T>&output) {
53          output.push_back(this->data);
54          if (this->left_child != nullptr)
            ↪  this->left_child->preorder_traversal(output);
55          if (this->right_child != nullptr)
            ↪  this->right_child->preorder_traversal(output);
56      }
57
58      void mirror_preorder_traversal(std::vector<T>&output) {
59          output.push_back(this->data);
60          if (this->right_child != nullptr)
            ↪  this->right_child->mirror_preorder_traversal(output);
61          if (this->left_child != nullptr)
            ↪  this->left_child->mirror_preorder_traversal(output);
```

```
62      }
63
64      void postorder_traversal(std::vector<T>&output) {
65          if (this->left_child != nullptr)
            ↪  this->left_child->postorder_traversal(output);
66          if (this->right_child != nullptr)
            ↪  this->right_child->postorder_traversal(output);
67          output.push_back(this->data);
68      }
69
70      void mirror_postorder_traversal(std::vector<T>&output) {
71          if (this->right_child != nullptr)
            ↪  this->right_child->mirror_postorder_traversal(output);
72          if (this->left_child != nullptr)
            ↪  this->left_child->mirror_postorder_traversal(output);
73          output.push_back(this->data);
74      }
75  };
76
77  int main() {
78      std::vector<int> origin_vector;
79      int temp;
80      while (std::cin >> temp) {
81          origin_vector.push_back(temp);
82      }
83      auto generator = origin_vector.cbegin();
84      BiSortNode<int> bi_sort_node{*generator++};
85      while (generator != origin_vector.cend()) {
86          bi_sort_node.insert(*generator++);
87      }
88
89      std::vector<int> preorder_traversal_result;
90      bi_sort_node.preorder_traversal(preorder_traversal_result);
91      std::vector<int> mirror_result;
92      bi_sort_node.mirror_preorder_traversal(mirror_result);
93
94      std::ofstream outfile;
95      outfile.open("temp_out.txt", std::ios::out);
96
97      if (origin_vector == preorder_traversal_result || origin_vector ==
        ↪  mirror_result) {
98          std::cout << "Yes" << std::endl;
```

```
        std::vector<int> post_result;
        bi_sort_node.postorder_traversal(post_result);
        for (const auto&i: post_result) {
            std::cout << i << " ";
        }
        std::cout << std::endl;
        std::vector<int> mirror_post_result;
        bi_sort_node.mirror_postorder_traversal(mirror_post_result);
        for (const auto&i: mirror_post_result) {
            outfile << i << " ";
        }
        outfile << std::endl;
    }
    else {
        std::cout << "No" << std::endl;
    }

    outfile.close();
    return 0;
}

// 输入:
// 41 40 19 8 5 5 17 32 39 32 98 95 68 56 60 59 67 94 77 98

// 输出:
// Yes
// 5 5 17 8 32 39 32 19 40 59 67 60 56 77 94 68 95 98 98 41

// temp_out.txt 文件输出:
// 98 77 94 67 59 60 56 68 95 98 32 39 32 17 5 5 8 19 40 41
```

2. 第七章作业 2.cpp

```
//
// Created by 423A35C7 on 2023-12-15.
//
// 09:19
// 20:06

#include <algorithm>
#include <cassert>
```

```cpp
#include <functional>
#include <iostream>
#include <memory>
#include <vector>

template<typename T>
class BiSortNode {
public:
    using NodePtr = std::unique_ptr<BiSortNode>;
    // 好像这里没法使用 initializer_list，initializer_list 是对于多个相同值的初始化，而不是一个东西的初始化参数列表
    template<typename... Args>
    explicit BiSortNode(Args... args, std::function<bool(T, T)> _compare) : data(args...) {
        this->compare_function_(_compare);
    }

    template<typename... Args>
    explicit BiSortNode(Args... args) : data(args...) {
        this->compare_function_ = [](T&a, T&b) { return a < b; };
    }

    void update_depth_and_balance_factor() {
        const int left_depth = this->left_child ? this->left_child->depth + 1 : 0;
        const int right_depth = this->right_child ? this->right_child->depth + 1 : 0;
        this->balance_factor = left_depth - right_depth;
        this->depth = std::max(left_depth, right_depth);
    }

    // 这里的左旋指的是 LL 型需要进行的旋转，名称不一定对
    // 这部分用注释不好解释，建议最好搜一下网上的图解，理解了之后再看这部分代码
    static void left_rotate(NodePtr&unbalanced) {
        NodePtr pivot = std::move(unbalanced->left_child);
        unbalanced->left_child = std::move(pivot->right_child);
        pivot->right_child = std::move(unbalanced);
        unbalanced = std::move(pivot);
        unbalanced->update_depth_and_balance_factor();
        unbalanced->right_child->update_depth_and_balance_factor();
    }

```

```
47      // 这里的右旋指的是 RR 型需要进行的旋转，名称不一定对
48      // 这部分用注释不好解释，建议最好搜一下网上的图解，理解了之后再看这部分代码
49      static void right_rotate(NodePtr&unbalanced) {
50          NodePtr pivot = std::move(unbalanced->right_child);
51          unbalanced->right_child = std::move(pivot->left_child);
52          pivot->left_child = std::move(unbalanced);
53          unbalanced = std::move(pivot);
54          unbalanced->update_depth_and_balance_factor();
55          unbalanced->left_child->update_depth_and_balance_factor();
56      }
57
58      static void insert(NodePtr&current_node, T new_data) {
59          // int balance_diff = 0;
60          if (new_data < current_node->data) {
61              // 如果新的节点比当前节点小就找它的左子树
62              if (current_node->left_child == nullptr) {
63                  current_node->left_child =
                    ↪  std::make_unique<BiSortNode>(new_data);
64              }
65              else {
66                  current_node->insert(current_node->left_child, new_data);
67              }
68          }
69          else {
70              // 否则找右子树
71              if (current_node->right_child == nullptr) {
72                  current_node->right_child =
                    ↪  std::make_unique<BiSortNode>(new_data);
73              }
74              else {
75                  current_node->insert(current_node->right_child, new_data);
76              }
77          }
78          current_node->update_depth_and_balance_factor();
79          if (current_node->balance_factor > 1) {
80              // 左子树比右子树高
81              assert(current_node->left_child->balance_factor != 0); // 此时
                ↪  左子树的平衡因子不应为 0
82              if (current_node->left_child->balance_factor < 0) // 符合 LR 型
83                  current_node->right_rotate(current_node->left_child); // 先
                    ↪  进行右旋（逆时针），转化为 LL
84              current_node->left_rotate(current_node); // 左旋（顺时针）
```

```
85          }
86          else if (current_node->balance_factor < -1) {
87              // 右子树比左子树高
88              assert(current_node->right_child->balance_factor != 0); // 此时
                ↪  右子树的平衡因子不应为 0
89              if (current_node->right_child->balance_factor > 0) // 符合 RL
                ↪  型
90                  current_node->left_rotate(current_node->right_child); // 先
                    ↪  进行左旋 (顺时针), 转化为 RR 型
91              current_node->right_rotate(current_node); // 右旋 (逆时针)
92          }
93          // current_node->balance_factor += balance_diff;
94          current_node->update_depth_and_balance_factor();
95          // return balance_diff;
96      }
97
98      void inorder_traversal(std::vector<T>&output) {
99          if (this->left_child != nullptr)
            ↪  this->left_child->inorder_traversal(output);
100         output.push_back(this->data);
101         if (this->right_child != nullptr)
            ↪  this->right_child->inorder_traversal(output);
102     }
103
104     void preorder_traversal(std::vector<T>&output) {
105         output.push_back(this->data);
106         if (this->left_child != nullptr)
            ↪  this->left_child->preorder_traversal(output);
107         if (this->right_child != nullptr)
            ↪  this->right_child->preorder_traversal(output);
108     }
109
110     void postorder_traversal(std::vector<T>&output) {
111         if (this->left_child != nullptr)
            ↪  this->left_child->postorder_traversal(output);
112         if (this->right_child != nullptr)
            ↪  this->right_child->postorder_traversal(output);
113         output.push_back(this->data);
114     }
115
116     void traverse_and_output(std::ostream&out,
        ↪  std::function<void(std::vector<T>&)> traverse_function) {
```

```
117         std::vector<T> traverse_result;
118         traverse_function(traverse_result);
119         for (auto const&i: traverse_result)
120             std::cout << i << " ";
121         std::cout << std::endl;
122     }
123 
124 private:
125     NodePtr left_child;
126     NodePtr right_child;
127     int balance_factor = 0; // 平衡因子
128     int depth = 0; // 树的深度（高度）
129     T data;
130     std::function<bool(T&, T&)> compare_function_;
131 };
132 
133 
134 int main() {
135     std::vector<int> origin_vector;
136     int temp;
137     while (std::cin >> temp) {
138         origin_vector.push_back(temp);
139     }
140     auto generator = origin_vector.cbegin();
141     BiSortNode<int>::NodePtr bi_sort_node{new
    ↪  BiSortNode<int>(*generator++)};
142     while (generator != origin_vector.cend()) {
143         bi_sort_node->insert(bi_sort_node, *generator++);
144     }
145     std::vector<int> traversal_result;
146     bi_sort_node->inorder_traversal(traversal_result);
147     std::cout << "为了便于验证，以下依次输出平衡二叉树的前序、中序、后序遍历
    ↪  结果: " << std::endl;
148     bi_sort_node->traverse_and_output(std::cout,
    ↪  [&bi_sort_node](std::vector<int>&output) {
149         bi_sort_node->preorder_traversal(output);
150     });
151     bi_sort_node->traverse_and_output(std::cout,
    ↪  [&bi_sort_node](std::vector<int>&output) {
152         bi_sort_node->inorder_traversal(output);
153     });
```

```
154     bi_sort_node->traverse_and_output(std::cout,
        ↪  [&bi_sort_node](std::vector<int>&output) {
155         bi_sort_node->postorder_traversal(output);
156     });
157     return 0;
158 }
159 
160 // 输入:
161 // 98 77 94 67 59 60 56 68 95 98 32 39 32 17 5 5 8 19 40 41
162 
163 // 输出:
164 // 为了便于验证，以下依次输出平衡二叉树的前序、中序、后序遍历结果:
165 // 67 39 17 5 5 8 32 19 32 59 41 40 56 60 94 77 68 98 95 98
166 // 5 5 8 17 19 32 32 39 40 41 56 59 60 67 68 77 94 95 98 98
167 // 5 8 5 19 32 32 17 40 56 41 60 59 39 68 77 95 98 98 94 67
```

3. CMakeLists.txt

```
1  cmake_minimum_required(VERSION 3.26)
2  project(chapter7)
3  
4  set(CMAKE_CXX_STANDARD 23)
5  
6  add_executable(binsort1 第七章作业 1.cpp)
7  add_executable(binsort2 第七章作业 2.cpp)
8  #set(EXECUTABLE_OUTPUT_PATH R:/) # 在 WSL 里不能这样
9  
10 #set(CMAKE_CXX_FLAGS_RELEASE -fexec-charset=GBK) # 在 WSL 里不需要这样了,
   ↪  Linux 的命令行默认编码就是 UTF-8
```